MLE 2022

4th International Workshop on Modeling Language Engineering
Co-located with MODELS 2022

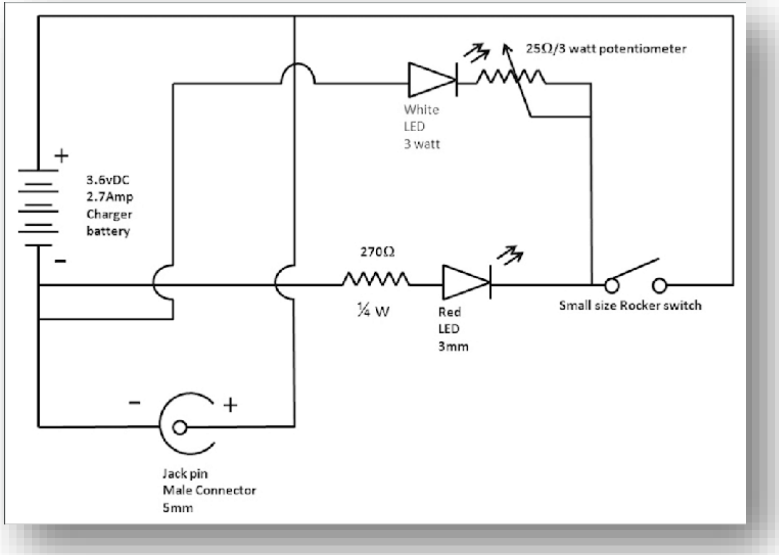
CLASSIFICATION AND MAPPING OF LAYOUT ALGORITHMS FOR USAGE IN GRAPH-LIKE MODELING LANGUAGES

Montreal, October 2022

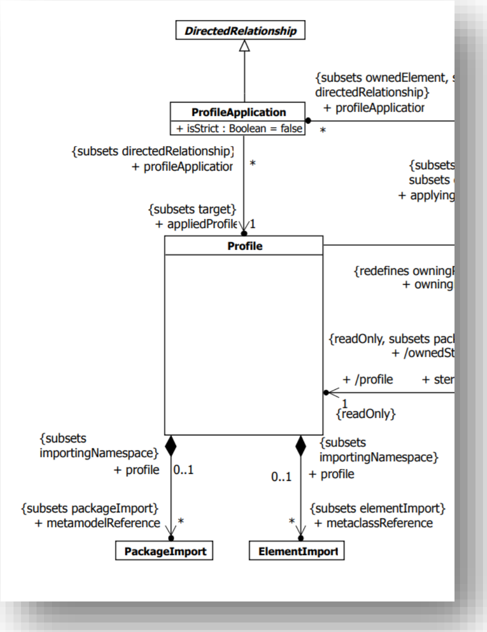
Gregor Wrobel, **Robert Scheffler**

Background

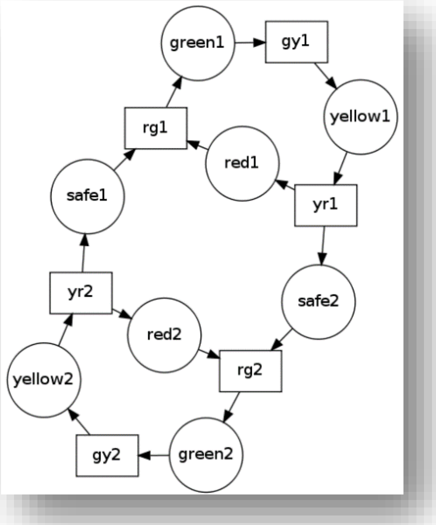
Technical Drawings – Modeling Languages – Graph Drawing



https://www.researchgate.net/figure/Circuit-diagram-for-the-battery-pack_fig2_272421945



https://www.omg.org/spec/UML/2.5.1/PDF



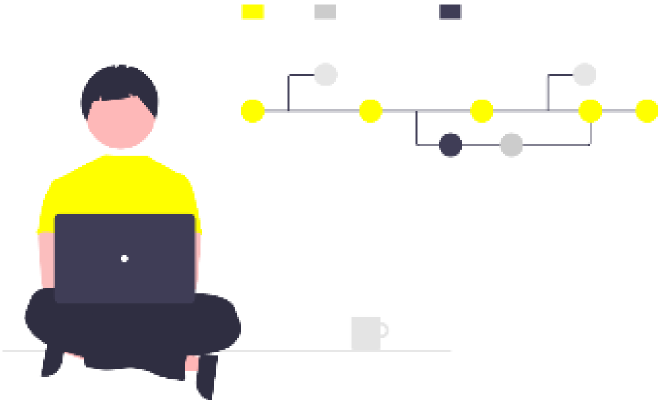
https://graphviz.org/Gallery/directed/traffic_lights.html

Motivation

Modelers and the Modeling Process

- GPML, e.g. UML
  - Loosely defined concrete syntax
  - Modeling experts
- Technical DSL, e.g. circuit diagrams
  - Strongly defined concrete syntax
  - Domain experts

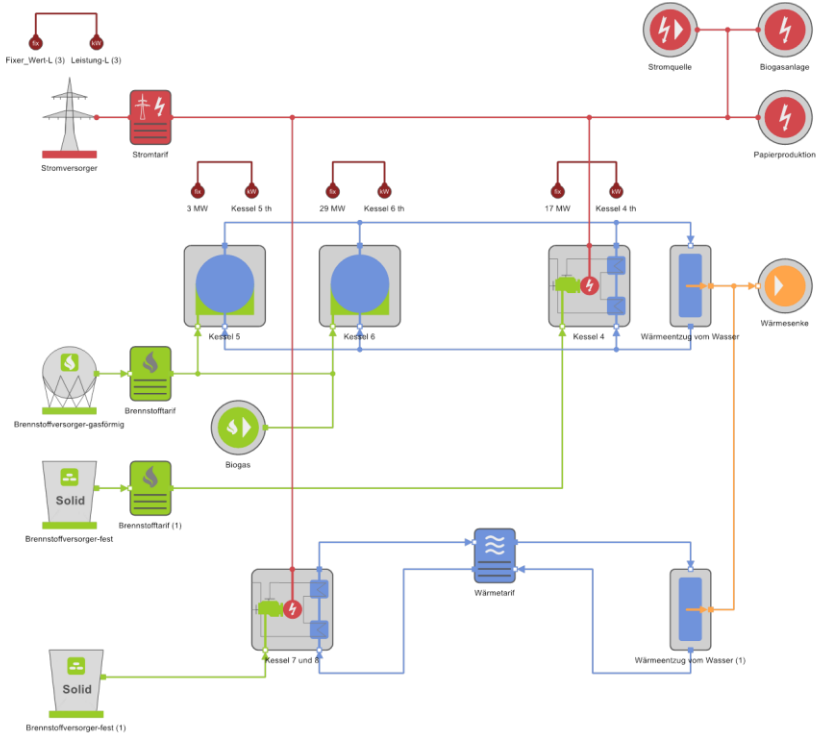
- Layout is important for
  - understanding,
  - creating,
  - and editing models



Motivation

Applied Graph Drawing

- Graph Drawing studies
  - Placement of vertices
  - Routing of edges
  - Types of graphs
- Challenges
  - Extending the classical graph model
  - Finding suitable layout algorithms for a given modeling language



Proposed Idea

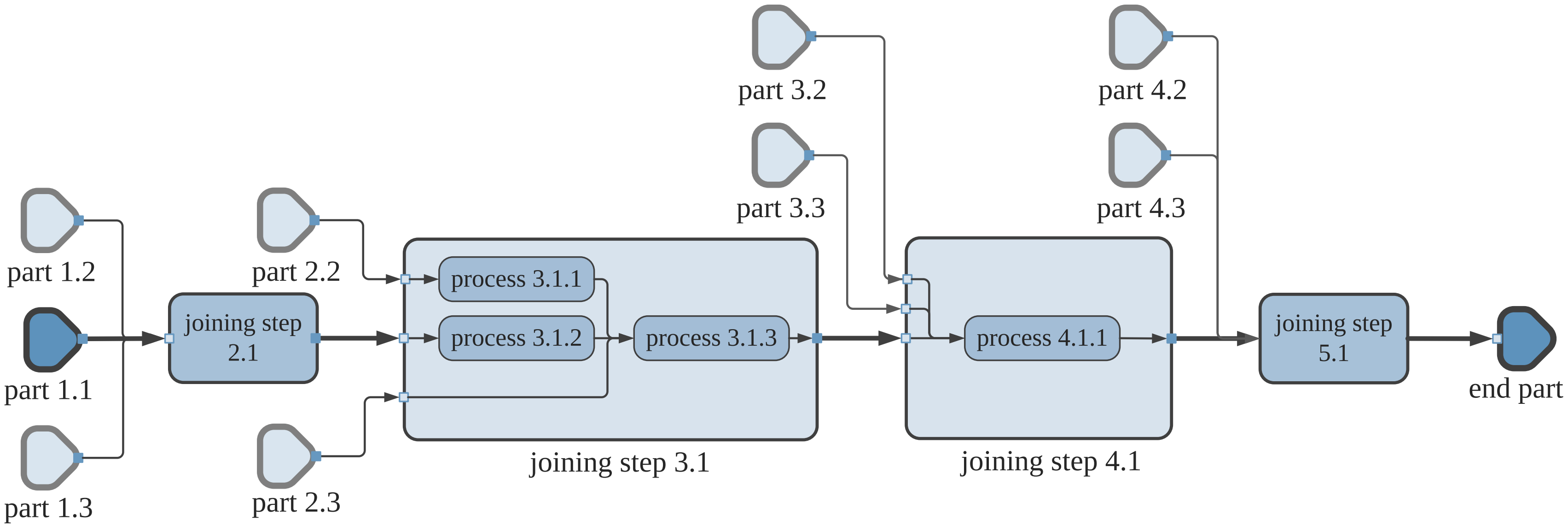
- Classification schemes for
  - Concrete syntax of GLML
  - Layout procedures
- Mapping between classifiers

- Developers can reuse existing layout procedures
  - e.g., in libraries like the Eclipse Layout Kernel
- Layout algorithms with parameters for different concrete syntax
  - Mapping between GLML and parameter sets

- Easier tool development

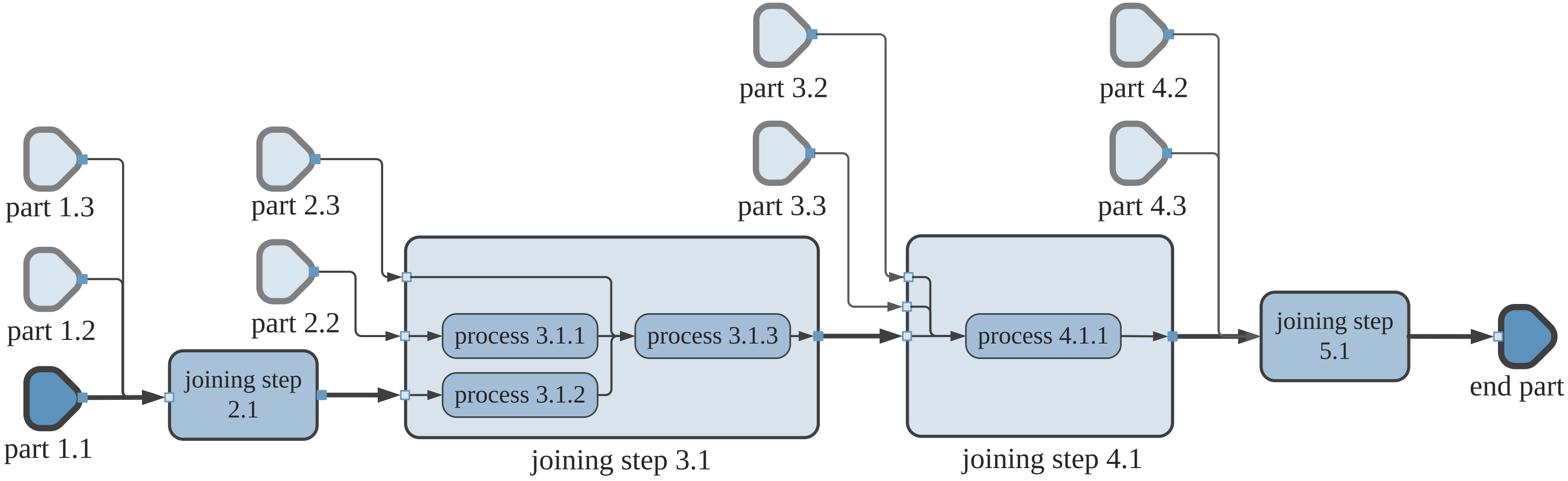
Example

Process Model



Example

Process Model



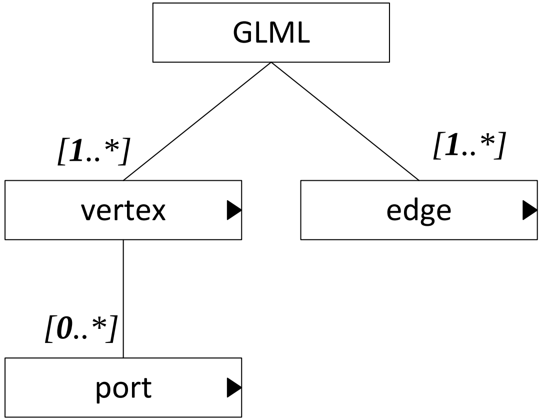
Classification Schemes

Feature Diagrams

- Classifiers for layout and GLML have corresponding features

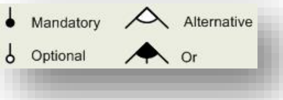
- Subfeatures
  - Vertex: label, ports, nesting, rotation, mirroring, placement
  - Port: label, position, direction, nested, valency
  - Edge: label, structure, direction, across nesting, routing

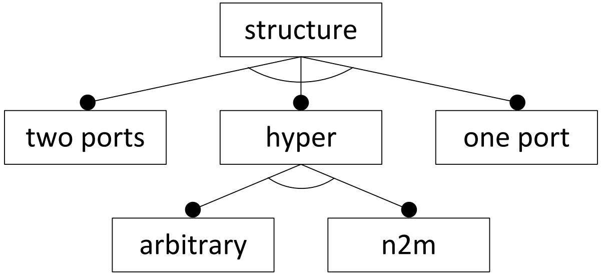
- Rendering aspects are excluded
  - Symbols, colors, …



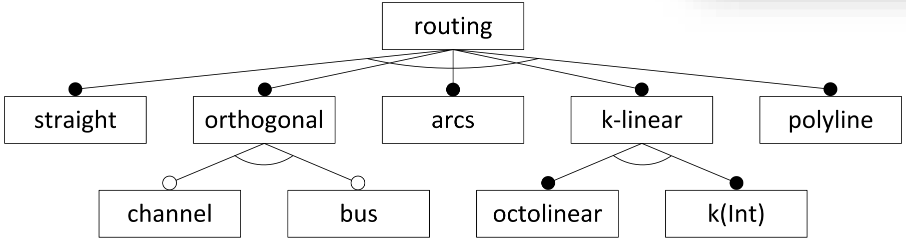
Classification Scheme

Snippet: Edge Classification

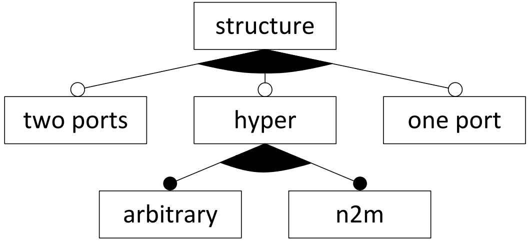




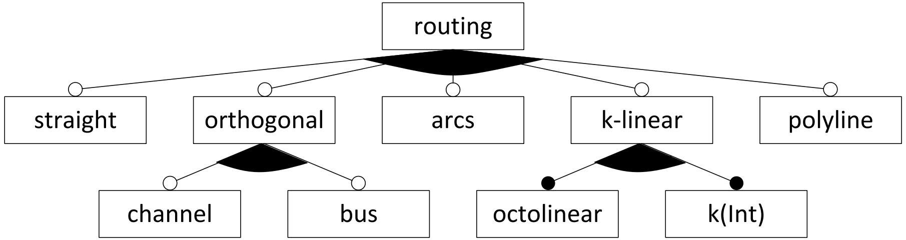
Structure Classifier for Edges in GLML



Routing Classifier for Edges in GLML



Structure classifier for edges in Layout Algorithm



Routing classifier for edges in Layout Algorithm

Classifier Mapping

Mapping Operator

For a given classification scheme *C,* a mapping between a specific GLML *G* and a concrete layout algorithm *L* is defined as:

A GLML *G* and a layout algorithm *L* can be mapped, or $G \equiv L = true$, if $G.f \equiv L.f = true$, $\forall f \in C.$

Conversely, a GLML *G* and a layout algorithm *L* cannot be mapped, or $G \equiv L = false$, if $G.f \equiv L.f = false, \exists f \in C.$

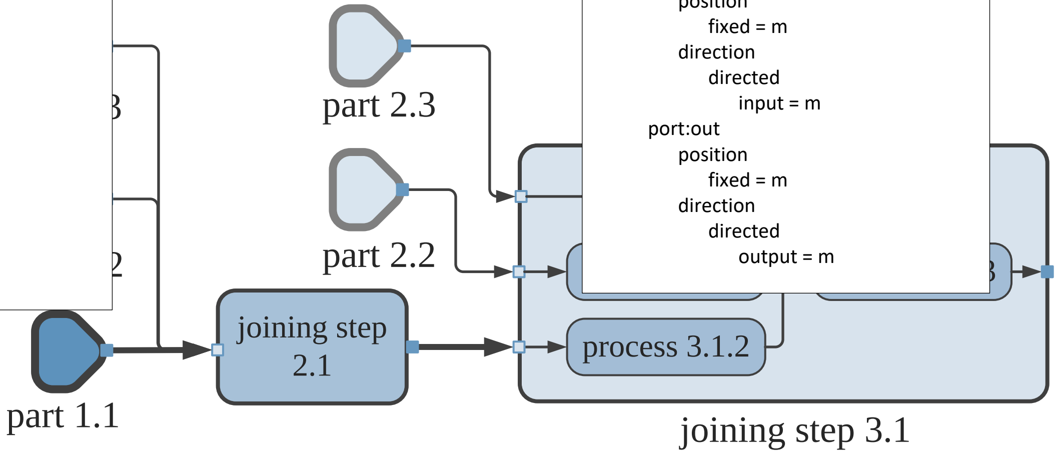
#	GLML.f	Layout.f	GLML.f ≡ Layout.f
1	*m*	*s*	*true*
2	*f*	*s*	*true*
3	$\neg m \wedge \neg f = \emptyset$	*s*	*true*
4	*m*	*n*	*false*
5	*f*	*n*	*true*
6	$\neg m \wedge \neg f = \emptyset$	*n*	*true*
7	*m*	*m*	*true*
8	*f*	*m*	*false*
9	$\neg m \wedge \neg f = \emptyset$	*m*	*false*
10	*m*	*f*	*false*
11	*f*	*f*	*true*
12	$\neg m \wedge \neg f = \emptyset$	*f*	*false*
13	*m*	$\neg n \wedge \neg s \wedge \neg m \wedge \neg f = \emptyset$	*true*
14	*f*	$\neg n \wedge \neg s \wedge \neg m \wedge \neg f = \emptyset$	*true*
15	$\neg m \wedge \neg f = \emptyset$	$\neg n \wedge \neg s \wedge \neg m \wedge \neg f = \emptyset$	*true*

m	=	mandatory
f	=	forbidden
s	=	supported
n	=	not supported

Mapping Example

Process Model and Moving Vertex

- Layout:Moving Vertex
  - vertex:any
    - port:any
      - position
        - free = s
        - side = s
        - fixed = s
  - edge:any
    - structure
      - two port = s
      - hyper = s
      - one port = n
    - routing
      - straight = n
      - orthogonal = m
        - channel = n
        - bus = s
      - arcs = n
      - k-linear = n
      - polyline = n



- GLML:Process Model
  - vertex:part
    - port:out
      - position
        - fixed = m
      - direction
        - directed
          - output = m
  - vertex:joining step (not expanded)
    - port:in
      - position
        - fixed = m
      - direction
        - directed
          - input = m
    - port:out
      - position
        - fixed = m
      - direction
        - directed
          - output = m

- GLML:Process Model
  - vertex:joining step (expanded)
    - port:in
      - position
        - side = m
          - east = m
          - west = f
          - north = f
          - south = f
      - direction
        - directed
          - input = m
    - port:out
      - position
        - fixed = m
      - direction
        - directed
          - output = m
  - edge:part to joining step (not expanded)
    - structure
      - hyper = m
        - arbitrary = m
    - routing
      - othogonal = m
        - bus = m
    - direction
      - directed = m

Mapping Example

Process Model and Moving Vertex

#		Process Model	Moving Vertex	≡
	port classifier			
1	position.free	*any*	*s*	*true*
2	position.side	*any*	*s*	*true*
3	position.fixed	*any*	*s*	*true*
4	directed.output	*any*	*Ø*	*true*
5	directed.input	*any*	*Ø*	*true*
6	side.east	*any*	*Ø*	*true*
7	side.west	*any*	*Ø*	*true*
8	side.north	*any*	*Ø*	*true*
9	side.south	*any*	*Ø*	*true*

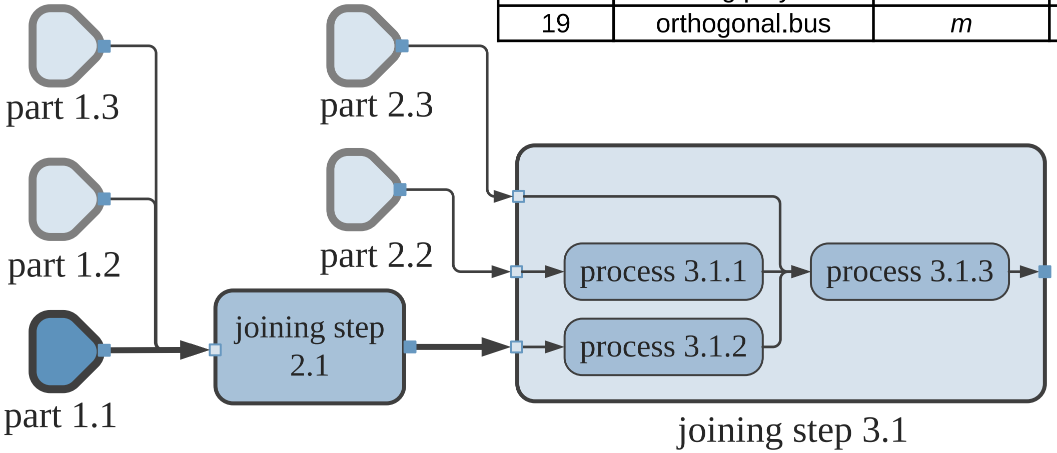
#		Process Model	Moving Vertex	≡
	edge classifier			
10	structure.two port	*any*	*s*	*true*
11	structure.hyper	*any*	*s*	*true*
12	structure.one port	*f*	*n*	*true*
13	hyper.arbitrary	*any*	*Ø*	*true*
14	routing.straight	*Ø*	*n*	*true*
15	routing.orthogonal	*m*	*m*	*true*
16	routing.arcs	*Ø*	*n*	*true*
17	routing.k-linear	*Ø*	*n*	*true*
18	routing.polyline	*Ø*	*n*	*true*
19	orthogonal.bus	*any*	*s*	*true*
20	orthogonal.channel	*Ø*	*n*	*true*
21	direction.directed	*any*	*Ø*	*true*

Mapping Example

Process Model and Graphviz Dot

#		Process Model	Graphviz Dot	≡
2	position.side	*m*	*n*	*false*
11	structure.hyper	*m*	*n*	*false*
15	routing.orthogonal	*m*	*s*	*true*
16	routing.arcs	*Ø*	*s*	*true*
18	routing.polyline	*Ø*	*s*	*true*
19	orthogonal.bus	*m*	*n*	*false*

```
Layout:ELK Graphviz Dot
    vertex:any
        port:any
            position
                free = s
                side = n
                fixed = s
    edge:any
        structure
            two port = s
            hyper = n
            one port = n
        routing
            straight = n
            orthogonal = s
                channel = n
                bus = n
            arcs = s
            k-linear = n
            polyline = s
```

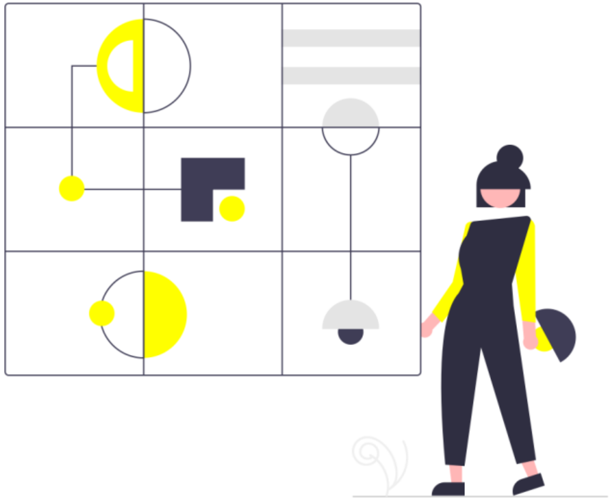


Conclusion

- Usability of GLMLs depends on good layout support
- Graph Drawing algorithms can be adapted for modeling languages

- Classification allows the automatic mapping of layout methods to GLML

- Better modeling tools for new languages
- Frameworks can offer well-fitted layouts
- Algorithm reuse is possible



Future Work

- Extending the classification scheme
  - Qualitative, e.g. *preferred* value
  - Quantitative, e.g. number of edges on a single port

- Integrating the mapping approach into a software framework
- Classification of existing languages (e.g. GPML)
- Application in new visualizations and editors

- Model-to-model transformations between GLML with distinct classifiers
  - Layout method reuse

Authors

We thank you for your attention!

Gregor Wrobel, wrobel@gfai.de
Robert Scheffler scheffler@gfai.de

Graph Based Engineering Systems,

Society for the Advancement of Applied Computer Science

The authors thank the German Research Foundation (DFG) for the financial support of the research project “Method for the Model-Driven Design of Deep Drawing Tools” (project number BA 6300/1-3).